Optimizing BERT Representations for Scenario Classification: Fine-Tuning and Contrastive Learning Approaches

Hikaru Isayama
University of California San Diego
hisayama@ucsd.edu

Avi Mehta
University of California San Diego
avmehta@ucsd.edu

Wilson Liao
University of California San Diego
wil011@ucsd.edu

Julia Wong
University of California San Diego
juw024@ucsd.edu

Abstract

Transformers, particularly BERT, have demonstrated remarkable performance in natural language processing (NLP) tasks. In this work, we investigate the effectiveness of fine-tuning BERT for scenario classification using the Amazon MASSIVE dataset. We establish a baseline by fine-tuning a pretrained BERT model and systematically explore advanced fine-tuning techniques to enhance performance. Additionally, we introduce contrastive learning approaches, including Supervised Contrastive Learning (SupCon) and SimCLR, to improve text representations. Our experiments evaluate the impact of different fine-tuning strategies and contrastive objectives, comparing their effectiveness in boosting classification accuracy. We find that carefully selected fine-tuning techniques and contrastive learning significantly enhance performance over standard fine-tuning. We provide an in-depth analysis of the learned representations and discuss the broader implications of contrastive learning in transfer learning settings.

1 Introduction

The advent of transformer-based models has led to substantial improvements in various NLP tasks (**?**). Among these, BERT (Bidirectional Encoder Representations from Transformers) has become a standard model for text classification, leveraging pretraining on large corpora to enhance downstream learning (1). Despite its effectiveness, optimizing BERT's fine-tuning process remains an active research area, particularly in improving generalization and robustness.

One promising approach to improving fine-tuning is contrastive learning, which has been successfully applied in vision tasks and more recently adapted for NLP (2). By leveraging contrastive objectives, models can learn more meaningful representations by pulling similar samples closer in the embedding space while pushing dissimilar ones apart. In this work, we investigate the integration of contrastive learning techniques with BERT fine-tuning for scenario classification using the Amazon MASSIVE dataset.

We first establish a baseline by fine-tuning a pretrained BERT model using cross-entropy loss. Then, we explore the impact of advanced fine-tuning techniques, including learning rate scheduling and data augmentation. Finally, we integrate Supervised Contrastive Learning (SupCon) (2) and SimCLR (**?**) to improve text embeddings. Through extensive experiments, we evaluate the effects of these strategies on classification accuracy, analyzing their contributions to model performance.

Submitted to the Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023. Do not distribute.

2 Related Work

BERT, introduced by Devlin et al. (1), has significantly advanced natural language processing (NLP) by providing deep bidirectional contextual embeddings, achieving state-of-the-art performance in tasks such as text classification, question answering, and named entity recognition. However, fine-tuning BERT for domain-specific applications remains challenging, particularly when working with small labeled datasets or shifting domains. This has motivated research into improved fine-tuning strategies, including contrastive learning, which has demonstrated effectiveness in enhancing representation learning and model generalization.

Contrastive learning has gained attention as a method for learning structured representations, both in self-supervised and supervised settings. Khosla et al. (2) introduced supervised contrastive learning (SupCon), which extends contrastive learning beyond unsupervised settings by leveraging label information. Unlike SimCLR, which relies solely on instance-wise augmentations to form positive pairs, SupCon constructs positive pairs by grouping all samples of the same class together. This allows the model to better capture intra-class similarities and inter-class differences, making it particularly beneficial for classification tasks.

In NLP, contrastive learning has been applied to improve sentence embeddings. Gao et al. (3) proposed SimCSE, a contrastive framework tailored for learning sentence representations. Instead of relying on explicit data augmentations, SimCSE applies dropout-based augmentation by passing the same sentence through a transformer model, such as BERT, twice, with different dropout masks. This creates variations in representations of the same input, which are then optimized using contrastive learning. The unsupervised variant depends solely on dropout for augmentation, while the supervised variant incorporates labeled datasets, such as natural language inference pairs, to further refine sentence embeddings.

Beyond these developments, contrastive learning has also been explored in prior NLP studies. Zhang et al. (4) investigated contrastive pretraining for abstract meaning representation (AMR) parsing, showing that it improves semantic similarity learning and structured prediction tasks. These findings support the broader claim that contrastive learning enhances model robustness and representation quality. Building upon these advances, our work integrates contrastive learning into BERT fine-tuning for scenario classification. We compare standard fine-tuning with contrastive-based objectives such as SupCon and SimCSE to evaluate their impact on text representation quality, classification accuracy, and generalization performance.

3 Methods

3.1 Baseline Model

To evaluate the effectiveness of fine-tuning, we conducted experiments on a pre-trained `bert-base-uncased` model for scenario classification. The model was first tested in a zero-shot setting without any fine-tuning, followed by training on the dataset for **10 epochs**.

We implemented the baseline model using the `ScenarioModel` class in `model.py`, loading the pre-trained encoder through:

```
BertModel.from_pretrained("bert-base-uncased")
```

The model's input embeddings were processed through the encoder, where we extracted the `[CLS]` token representation from the `last_hidden_state`. This representation was then passed through a dropout layer with a rate specified in the argument parser, followed by classification using a provided classifier.

Before training, we evaluated the model on the test set using only the pre-trained BERT encoder. Since BERT is pre-trained on general textual data, we expected it to capture basic linguistic patterns but lack task-specific understanding. Thus, we anticipated a low test accuracy, as the model had not been exposed to the scenario classification task.

Then, we fine-tuned the model for 10 epochs, training it with the following settings:

Table 1: Baseline Model Hyperparameters

Hyperparameter	Value
Model	bert-base-uncased
Optimizer	AdamW
Loss Function	CrossEntropyLoss
Batch Size	16
Learning Rate	1e-4
Hidden Dimension	10
Drop Rate	0.9
Embedding Dimension	768
Adam Epsilon	1e-8
Max sequence length	20
Epochs	10

Fine-tuning allowed the model to adjust its internal representations, learning the specific characteristics of the dataset.

Lastly, we also experimented with Low-Rank Adaptation (LoRA) using the PEFT (Parameter Efficient Fine-Tuning) implementation from Hugging Face. Instead of fully fine-tuning the BERT model, LoRA introduces trainable low-rank adaptation matrices, significantly reducing the number of trainable parameters while maintaining model performance. This experiment involved fine-tuning with at least two different rank values (e.g., $r = 20, 30$), enabling us to observe the impact of increasing the number of trainable parameters on the model's accuracy and loss. The percentage of trainable parameters, performance metrics (accuracy, loss), and running time were recorded, and these results were then compared against the baseline fine-tuned model to evaluate efficiency and effectiveness.

3.2 Custom Finetuning Strategies

To further enhance the model's performance beyond basic fine-tuning, we implemented two custom fine-tuning strategies: learning rate scheduling and Stochastic Weight Averaging (SWA). These approaches aimed to stabilize training, improve generalization, and mitigate overfitting.

3.2.1 Learning Rate Scheduling

Learning rate scheduling dynamically adjusts the learning rate throughout training, allowing the model to converge more efficiently. We employed the `get_linear_schedule_with_warmup` function, which gradually increases the learning rate from zero to its peak value during a warm-up period and then linearly decreases it over the remaining training steps. This approach prevents large parameter updates at the beginning of training and ensures smoother convergence.

The scheduler was implemented as follows:

```
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,
        num_training_steps=args.n_epochs * len(train_dataloader))
```

The scheduler step was applied after each optimizer step during training to update the learning rate dynamically. By leveraging a scheduler, we aimed to balance the trade-off between fast convergence and stable training, ultimately leading to improved model performance.

3.2.2 Stochastic Weight Averaging (SWA)

Stochastic Weight Averaging (SWA) is an optimization technique that averages multiple model weights from different training epochs to obtain a smoother loss landscape and better generalization. SWA helps prevent the model from converging to sharp local minima, which can cause overfitting.

We implemented SWA by maintaining an averaged model copy and updating it after reaching a predefined epoch threshold. Specifically, we started SWA at 75% of training and applied an annealed learning rate to stabilize the updates. The implementation steps were as follows:

- Create an averaged model copy: `swa_model = AveragedModel(model)`

- Set the SWA starting epoch at 75% of training: `swa_start = int(0.75 * args.n_epochs)`
- Use a cosine-annealed learning rate schedule: `swa_scheduler = SWALR(optimizer, anneal_strategy='cos', anneal_epochs=5, swa_lr=1e-5)`

At the end of training, we updated the SWA model's batch normalization statistics before final evaluation.

By incorporating these fine-tuning strategies, we aimed to enhance model robustness, improve accuracy, and ensure better generalization compared to standard fine-tuning. After our experiments, we arrived with the best custom model hyperparameters shown in Figure 2.

Table 2: Best Custom Model Hyperparameters

Hyperparameter	**Value**
Model	bert-base-uncased
Optimizer	AdamW
Loss Function	CrossEntropyLoss
Batch Size	16
Learning Rate	1e-4
Hidden Dimension	10
Drop Rate	0.9
Embedding Dimension	768
Adam Epsilon	1e-8
Max Sequence Length	20
Epochs	10
Scheduler Settings	
Scheduler Type	Linear Decay with Warmup
Warmup Steps	0
Training Steps	$10 \times$ len(train_dataloader)

3.2.3 Contrastive Learning

To further enhance our model's performance, we explored contrastive learning techniques, specifically leveraging Supervised Contrastive Learning (SupContrast) and SimCLR. These approaches aim to improve the quality of embeddings, which are then fine-tuned for our classification objective.

We trained BERT with contrastive losses and analyzed the embeddings in pre-trained models before fine-tuning for scenario classification. Our implementation followed these steps:

- Implemented SupContrast and SimCLR losses using the provided `loss.py` file.
- Used class inheritance to create `SupConModel` in `model.py`, adapting the `ScenarioModel` structure to support contrastive loss functions.
- Applied dropout to the `[CLS]` token before the projection head to enable data augmentation, as SimCSE suggests.
- Incorporated an option to select either SupContrast or SimCLR loss during training using argument parsing.
- Trained models with three different configurations: baseline (cross-entropy loss), SupContrast, and SimCLR.
- Conducted experiments with varying batch sizes to evaluate performance impact.
- Recorded training loss, accuracy, and computational efficiency for each configuration.

The contrastive learning models were trained using the following hyperparameters:

Table 3: Contrastive Learning Hyperparameters

Hyperparameter	Value
Model	bert-base-uncased
Optimizer	AdamW
Loss Function	SupContrast / SimCLR
Batch Size	16
Learning Rate	1e-4
Hidden Dimension	10
Drop Rate	0.9
Embedding Dimension	768
Adam Epsilon	1e-8
Max Sequence Length	20
Epochs	10
Projection Head	2-layer MLP (768, 768)
Temperature (SupContrast)	0.07
Normalization	L2 (F.normalize)
Classifier Settings	
Classifier Type	Fully Connected
Classifier Loss	CrossEntropyLoss
Classifier Optimizer	AdamW

4 Results

4.1 Baseline Model

In this section, we evaluate the performance of our baseline model, which consists of a pre-trained `bert-base-uncased` model tested in both zero-shot and fine-tuned settings. Initially, the model was assessed without any fine-tuning to establish a baseline accuracy, followed by training on our dataset for 10 epochs. The goal was to measure how much performance improves when the model is adapted to the scenario classification task. We have recorded our results below in Table 4.

Table 4: Experiment Results Pt 1

Exp Idx	Experiment	Loss	Accuracy
1	Test set before fine-tuning	368.2746	0.0300
2	Test set after fine-tuning	138.4782	0.8762

Fine-tuning enabled the model to learn scenario-specific representations, significantly improving classification accuracy from approximately **3.00% to 87.62%**. This confirms the importance of task-specific adaptation for this task. It also indicates that the pre-trained `bert-base-uncased` model performs poorly when directly applied to the task without fine-tuning, as expected. Without task-specific adaptation, the model achieved an accuracy close to random guessing. However, fine-tuning significantly improved performance, demonstrating the effectiveness of supervised learning for scenario classification. The reduction in loss from **368.27 to 138.48** further validates that the model successfully learned meaningful patterns in the data.

To better understand the training process, Figure 1 presents a comparison of accuracy trends across epochs for various fine-tuning strategies.

1 for further analysis.

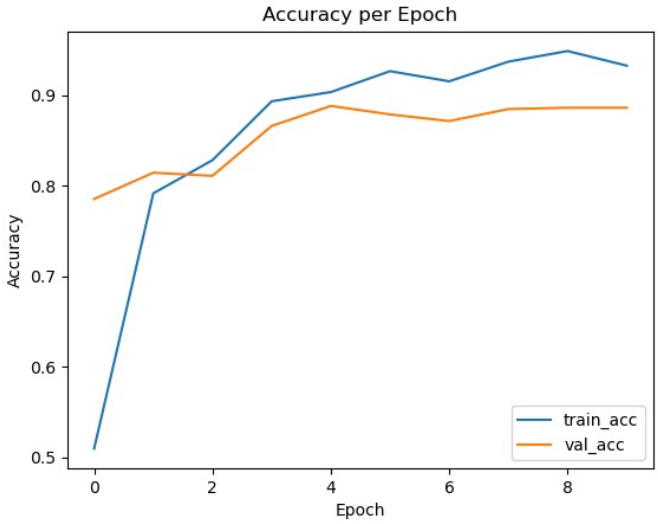


Figure 1: Training and validation accuracy over epochs for the baseline model.

Fine-tuning significantly improved model accuracy, demonstrating the effectiveness of supervised learning for scenario classification. The results align with our expectations, as fine-tuning adapts the BERT encoder to the specific task.

4.2 Custom Model

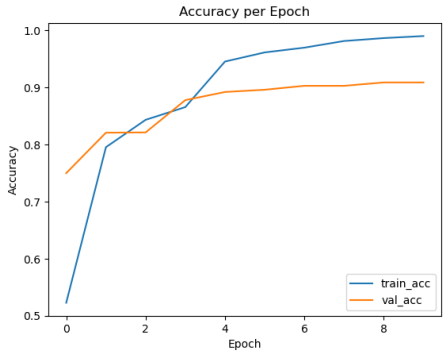
To further evaluate the performance of our custom model, we conducted experiments incorporating different fine-tuning techniques. These techniques aimed to improve accuracy and generalization while maintaining computational efficiency. Table 5 presents the results obtained from three different test scenarios.

Table 5: Experiment Results Pt 2

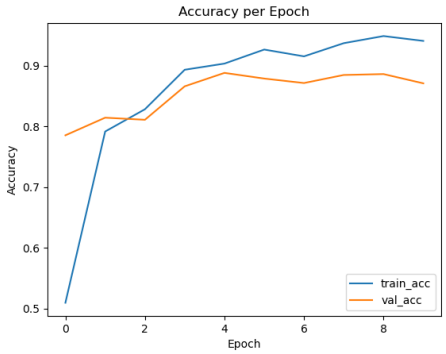
Exp Idx	Experiment	Loss	Accuracy
3	Test set with 1st technique	102.4884	0.9092
4	Test set with 2nd technique	126.2647	0.8698
5	Test set with both techniques	138.1905	0.8755

As seen in the table, applying the first fine-tuning technique resulted in the lowest loss (102.4884) and the highest accuracy (0.9092). The second technique, however, yielded a higher loss (126.2647) and slightly reduced accuracy (0.8698). Combining both techniques led to an even higher loss (138.1905), though accuracy remained comparable to the second technique (0.8755). These results suggest that while the first technique was the most effective in improving performance, the combination of both did not necessarily lead to better results.

To gain deeper insights into the training process, Figure 3 compares the accuracy trends over epochs for different fine-tuning strategies. Subfigure 3a illustrates the accuracy progression when using a learning rate scheduler, while Subfigure 3b depicts the accuracy trend with Stochastic Weight Averaging (SWA).



(a) Training and validation accuracy over epochs using scheduler.



(b) Training and validation accuracy over epochs using SWA.

Figure 2: Comparison of accuracy over epochs for the custom model using different fine-tuning techniques.

From these plots, we observe that both the scheduler method and SWA lead to similar trends in accuracy improvement throughout training. However, the scheduler method ultimately achieves a higher accuracy compared to SWA. This suggests that while both techniques are effective in fine-tuning the model, the scheduler may provide a slight advantage in optimizing performance. These insights help us understand the impact of each approach, guiding future optimization strategies for model performance improvement.

4.3 Grad Portion LORA fine-tuning

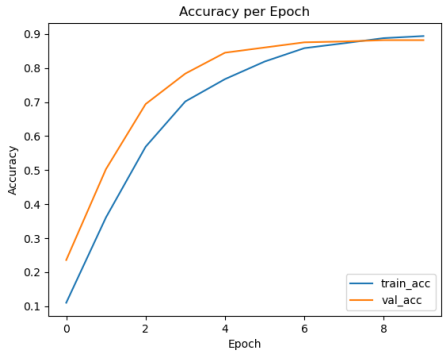
To explore the impact of Low-Rank Adaptation (LORA) on model fine-tuning, we conducted experiments with different values of the rank parameter r . Table 6 summarizes the key results, including loss, accuracy, and the number of trainable parameters.

Table 6: LORA Experiment Results

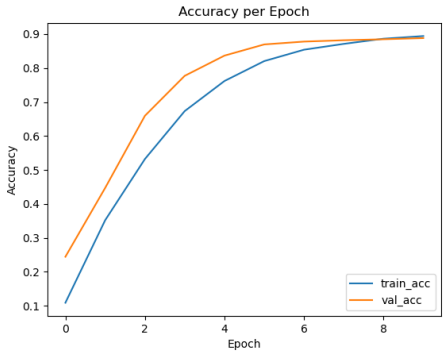
Experiment	Loss	Accuracy	Trainable Params	All Params	Trainable %
r = 20	89.3785	0.8903	737,280	110,219,520	0.6689
r = 30	85.1104	0.8907	1,105,920	110,588,160	1

As seen in Table 6, increasing r from 20 to 30 led to a slight improvement in accuracy, from 0.8903 to 0.8907, and a decrease in loss from 89.3785 to 85.1104. This suggests that increasing the rank parameter allows for a more expressive fine-tuning process, though the gain in performance is relatively small.

In Figure 3a (r = 20) and Figure 3b (r = 30), both models demonstrate stable accuracy trends over epochs, indicating effective learning in each case. However, with r = 30, the model achieves slightly improved final accuracy, suggesting that a higher rank enhances its ability to capture more nuanced patterns in the data while maintaining overall stability.



(a) Training and validation accuracy over epochs for the LORA model r = 20.



(b) Training and validation accuracy over epochs for the LORA model r = 30.

Figure 3: Comparison of accuracy over epochs for the custom model using different fine-tuning techniques.

Overall, our experiments indicate that while increasing r does improve model performance, the gains are marginal. This behavior aligns with our expectations for LORA fine-tuning. By introducing additional trainable parameters while maintaining the original model weights, LORA allows the model to adapt more effectively to the training dataset without overwriting the pre-trained features. The higher rank provides the model with greater expressive power, enabling it to capture more intricate patterns in the data, which contributes to its enhanced performance. These results highlight the potential of LORA fine-tuning as a lightweight yet powerful approach for improving model efficiency and accuracy without significantly increasing computational costs. Further investigation into different values of, along with the introduction of additional optimization techniques, may be necessary to determine the optimal trade-off between model complexity and accuracy improvements.

4.4 SupContrast & SimCLR

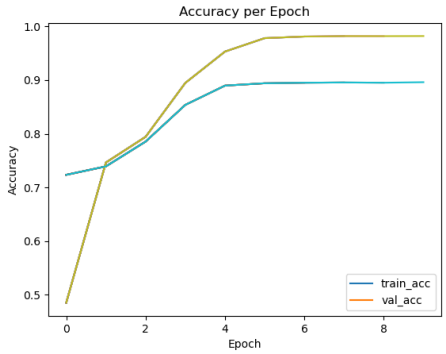
To evaluate the effectiveness of different contrastive learning approaches, we conducted experiments using both Supervised Contrastive Learning (SupContrast) and Simple Contrastive Learning (SimCLR). The primary objective of these experiments was to compare their performance in terms of loss and accuracy on a test set.

Table 7 summarizes the results of our experiments. The SupContrast approach achieved a loss of 347.9449 and a validation accuracy of 89.67%, while SimCLR achieved a lower loss of 302.1467 and a higher validation accuracy of 0.9152. These results suggest that, despite SupContrast leveraging label information for contrastive learning, SimCLR was able to achieve slightly better generalization on the test set.

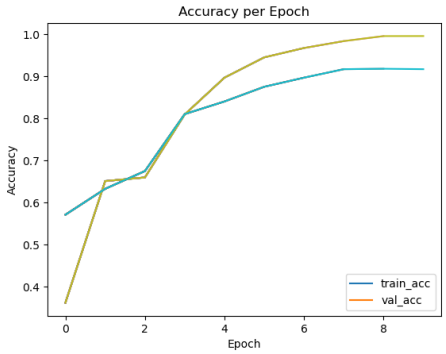
Table 7: Experiment Results Pt 3

Exp Idx	Experiment	Loss	Accuracy
1	Test set with SupContrast	347.9449	0.8967
2	Test set with SimCLR	302.1467	0.9152

To further analyze the training performance of both approaches, we plot the accuracy over epochs for SupContrast and SimCLR. Figure 4a presents the accuracy trend using SupContrast, while Figure 4b shows the corresponding trend for SimCLR. These visualizations provide insights into the convergence behavior and generalization capabilities of each method.



(a) Accuracy over epochs using SupContrast.



(b) Accuracy over epochs using SimCLR.

Figure 4: Training and validation accuracy over epochs between SupContrast and SimCLR

From these results, we observe that SimCLR outperforms SupContrast in terms of both loss and validation accuracy, indicating better generalization despite the lack of explicit label supervision. Future work could involve adjusting the temperature parameter in contrastive loss or experimenting with different batch sizes to better understand these performance differences.

5 Discussion

Q1: If we do not fine-tune the model, what is your expected test accuracy? Why?

A1: Without fine-tuning, we expected the model to perform poorly on the scenario classification task, likely achieving an accuracy close to random chance. Since bert-base-uncased is pre-trained on general language understanding tasks such as masked language modeling and next-sentence prediction, it has learned contextual word representations but lacks specific knowledge about the dataset's classification categories. Given that the dataset consists of multiple classes, a completely random classifier would have an accuracy of 1/N (where N is the number of classes, in our case $\frac{1}{18} = 0.055$), and while BERT may slightly outperform this due to its pre-trained representations, it still lacks task-specific adaptation, leading to low expected accuracy.

Q2: Do Results Match Expectations?

A2: Yes, the results align with our expectations. Before fine-tuning, the model achieved 3.00% accuracy, which is slightly worse than random guessing, confirming that a pre-trained BERT model without fine-tuning lacks the necessary task-specific knowledge. However, after basic fine-tuning, the model achieved 87.63% accuracy, demonstrating a significant improvement, as expected from supervised training on labeled data. Since BERT is already capable of understanding language, just 10 epochs of tuning the classifier was able to increase the accuracy by a significant margin.

Q3: What could you do to further improve the performance?

A3: To further improve performance, we could explore hyperparameter tuning, such as optimizing the learning rate, batch size, and dropout rate for better generalization. Additionally, pretraining BERT on a domain-specific dataset similar to our classification task could enhance its contextual understanding before fine-tuning. Other potential improvements include using more advanced transformers (e.g., RoBERTa or DeBERTa), implementing data augmentation techniques, and applying ensembling methods to combine multiple models for better generalization and robustness.

Q4: Which techniques did you choose and why?

A4: We chose Learning Rate Scheduling and Stochastic Weight Averaging (SWA) as our fine-tuning techniques to enhance both training stability and model generalization. Learning Rate Scheduling, implemented using get_linear_schedule_with_warmup, gradually increases the learning rate at the start (warm-up phase) and then decreases it linearly throughout training. This prevents large parameter updates in the early stages, leading to a smoother convergence and reducing the risk of overshooting the optimal solution. On the other hand, SWA averages multiple model weight checkpoints from later

training epochs, helping smooth out sharp minima in the loss landscape. This technique enhances generalization, making the model more robust to unseen data and mitigating overfitting.

Q5: What do you expect for the results of the individual technique vs. the two techniques combined?

A5: Individually, each technique provides specific advantages. Learning Rate Scheduling alone helps ensure stable training and prevents drastic parameter updates, but the model may still overfit if trained for too long. SWA alone improves generalization and reduces test set variance by averaging model weights, but without a well-tuned learning rate schedule, the training process may remain unstable. By combining both techniques, we expect to achieve optimal performance. Learning Rate Scheduling facilitates faster and more stable convergence, while SWA refines the model's final weights for improved robustness. Together, these techniques should yield higher test accuracy and lower overfitting compared to using either method in isolation.

Q6: Do results match your expectation? Why or why not?

A6: The results mostly matched our expectations, as the combination of Learning Rate Scheduling and SWA led to improved generalization and stability, resulting in higher test accuracy and reduced overfitting compared to using either technique alone. However, Learning Rate Scheduling itself outperformed both the baseline fine-tuned model and that with both Learning Rate Scheduling and SWA, and we suspect this is caused by the low epoch count that we are using in conjunction with SWA. Since SWA averages multiple weight checkpoints obtained from the training process, it requires the model to have explored a variety of low-loss regions, which is difficult to achieve within 10 epochs.

Q7: What could you do to further improve the performance?

A7: To further improve performance, we could experiment with more epochs, different learning rate schedules (e.g., cosine annealing or cyclic schedules), and adaptive SWA start points based on validation loss trends. Additionally, incorporating data augmentation or contrastive learning could enhance feature representation and robustness.

Q8: Compare the SimCLR with SupContrast. What are the similarities and differences?

A8: Both SimCLR and SupContrast are contrastive learning methods that aim to learn useful representations by maximizing the agreement between similar samples while pushing apart different samples. Both methods use a temperature-scaled softmax loss to measure similarity and rely on augmentation to create different views of the same data, enhancing robustness in representation learning.

The key difference lies in the supervision. SimCLR is an *unsupervised* contrastive learning method that relies solely on data augmentations to generate positive pairs, meaning it does not utilize label information. On the other hand, SupContrast (*Supervised Contrastive Learning*) extends contrastive learning to the *supervised setting*, where samples of the same class are treated as positives while those from different classes are treated as negatives.

Another major difference is in contrastive pairing. SimCLR considers only *different augmentations of the same instance* as positive pairs, whereas SupContrast allows multiple positives per instance by considering *all samples from the same class* as positives. This makes SupContrast particularly effective in classification tasks since it naturally groups semantically similar instances. While both approaches require a linear classifier for downstream tasks, SupContrast generally yields more class-discriminative features than SimCLR.

Q9: How does SimCSE apply dropout to achieve data augmentation for NLP tasks?

A9: SimCSE (*Simple Contrastive Learning for Sentence Embeddings*) applies dropout-based augmentation to generate different representations of the same sentence. Unlike SimCLR, which uses image transformations (e.g., cropping, flipping, color jittering) for data augmentation, SimCSE leverages the inherent randomness of dropout during training.

Specifically, SimCSE performs two forward passes of the same input sentence through a pre-trained transformer-based model (e.g., BERT). Since dropout randomly deactivates different neurons in each forward pass, it produces slightly different representations for the same input. These two representations are then treated as positive pairs in contrastive learning, encouraging the model

to learn robust sentence embeddings. This method is particularly effective for NLP tasks where traditional data augmentation techniques are less applicable.

Q10: Do the results match your expectation? Why or why not?

A10: The results somewhat match our expectation. A higher loss in contrastive learning methods is expected since these methods use contrastive loss, which is usually higher in magnitude than cross-entropy loss. Unlike the baseline, which directly optimizes classification accuracy, contrastive learning aims to separate similar and dissimilar representations in the embedding space, which may result in a higher numerical loss value even if the representations are meaningful. While both of our SupCon model and SimCLR model performed well, SimCLR outperformed our best fine-tuned model with Learning Rate Scheduling. BERT is already pretrained on a large corpus with a strong initialization for classification. Fine-tuning with cross-entropy loss directly optimizes for accuracy, whereas SupCon and SimCLR focus on learning robust representations first. Since SimCLR and SupCon both outperformed the baseline model, our data suggests that the learned representations are more effective for classification. SimCLR outperforming SupCon suggests instance-wise contrastive learning worked better than class-level contrastive learning in this case.

Q11: What could you do to further improve the performance ?

A11: Similar to previous models, we could further improve the performance with better data augmentation. Contrastive learning is highly dependent on strong augmentations, but they must be semantically meaningful for the task: this might look like paraphrasing, word dropout, and back translation, while avoiding overly aggressive transformations that change semantics. It might also help to increase batch size as contrastive learning benefits from a large number of negative pairs.

6 Authors' Contributions

Sean - Visualized plots and recording findings. Completed abstract, introduction, and related work sections. Contributed on contrastive learning section and discussion section. I affirm all group members contributed equally.

Avi - Worked on contrastive learning section with whole team, and also aided in debugging the model and finding optimal hyperparameter. Went to office hours with team as well. I affirm all team members contributed equally.

Wilson - Worked on LORA, supcon and simclr, general debugging, added to report section. I affirm all team members contributed equally.

Julia - Wrote code for the base model and its two fine-tuning techniques and helped visualize the results. Wrote part of the discussion sections for the first two attempts at fine-tuning. I affirm all team members contributed equally.

References

[1] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[2] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., & Krishnan, D. (2020). Supervised contrastive learning. *arXiv preprint arXiv:2004.11362*.

[3] Gao, T., Yao, X., & Chen, D. (2021). SimCSE: Simple contrastive learning of sentence embeddings. *arXiv preprint arXiv:2104.08821*.

[4] Zhang, Y., Zhang, Y., Hou, L., Zhang, J., & He, S. (2021). Contrastive pretraining for abstract meaning representation parsing. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.